Business Process Management

Workflow and Data Patterns: A formal semantics

Frank Puhlmann
Business Process Technology Group
Hasso Plattner Institut
Potsdam, Germany

Hasso Plattner Institut
IT Systems Engineering | Universität Potsdam

Foundations

- The Formalization of Workflow Patterns is based on ECA rules

ECA Rules

- ECA rules from active databases:
  - (on) Event,
  - (if) Condition,
  - (then) Action
- Different Coupling Modes
- Different Triggers

ON inserting a row in course registration table

IF over course capacity

THEN abort registration transaction

Example: ECA rule

ON inserting a row in course registration table

IF over course capacity

THEN notify registrar about unmet demands

ON inserting a row in course registration table

IF over course capacity

THEN put on waiting list

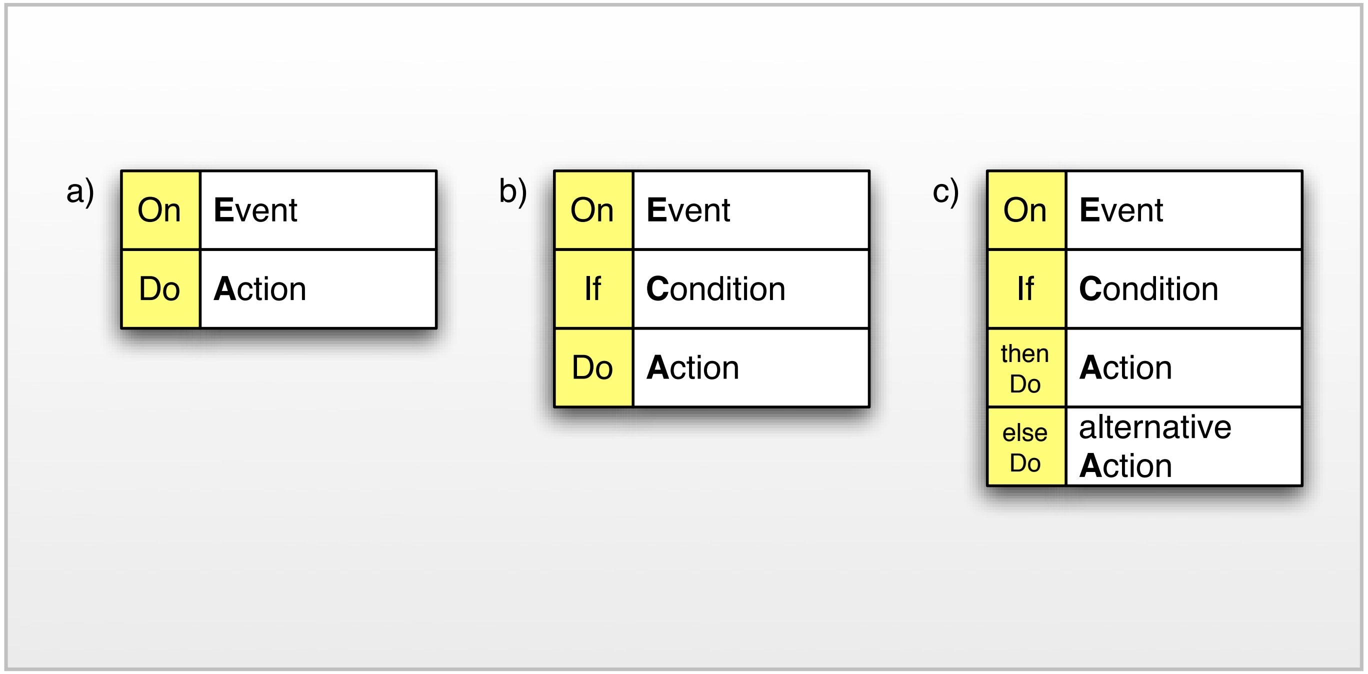
Example: ECA Conflicts

```
CREATE TRIGGER LimitSalaryRaise
    AFTER UPDATE OF Salary ON Employee
    REFERENCING OLD AS 0, NEW AS N
    FOR EACH ROW
    WHEN (N.Salary - O.Salary > 0.05*O.Salary)
        UPDATE Employee
        SET Salary = 1.05 * O.Salary
        Where Id = O.Id
```

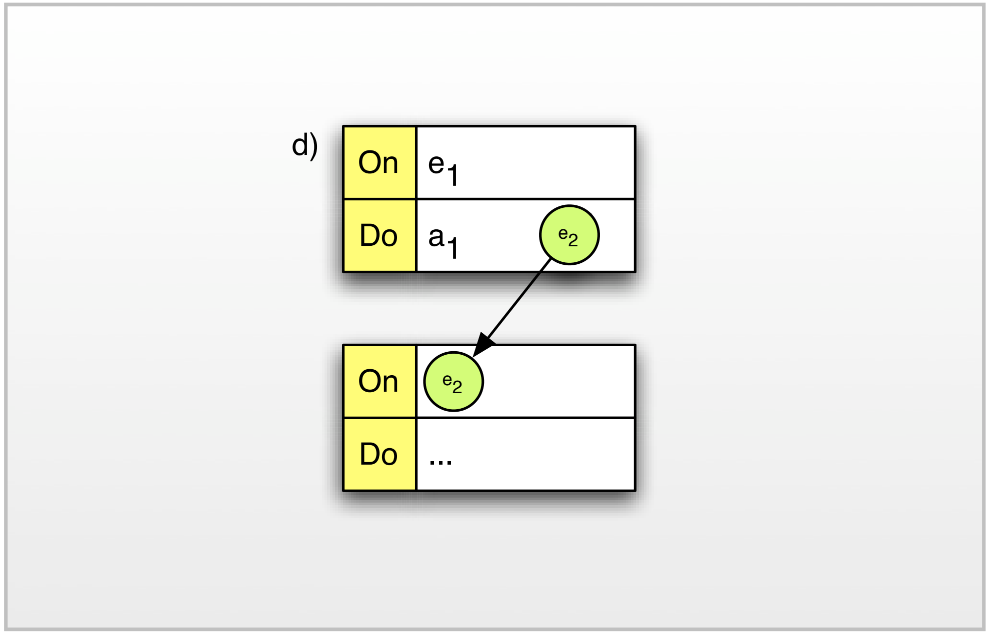
Business Rule Enforced with AFTER trigger

Event-based Routing

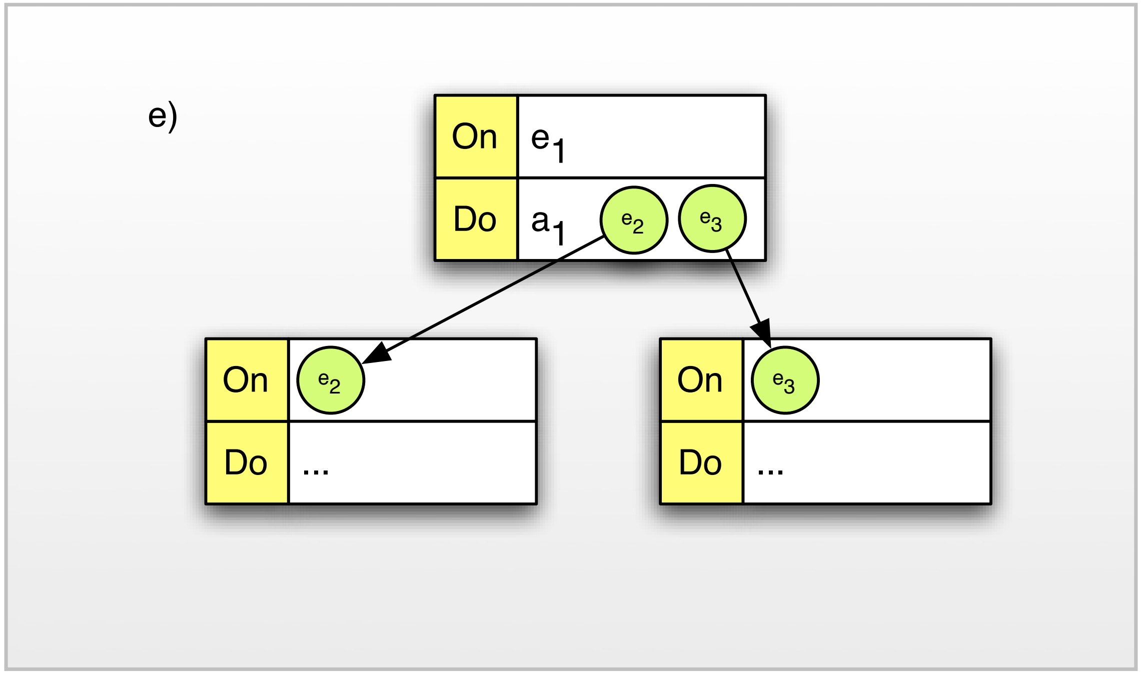
- The ECA approach has been adapted to workflows:
  - 1 Event
  - m Conditions
  - n Actions



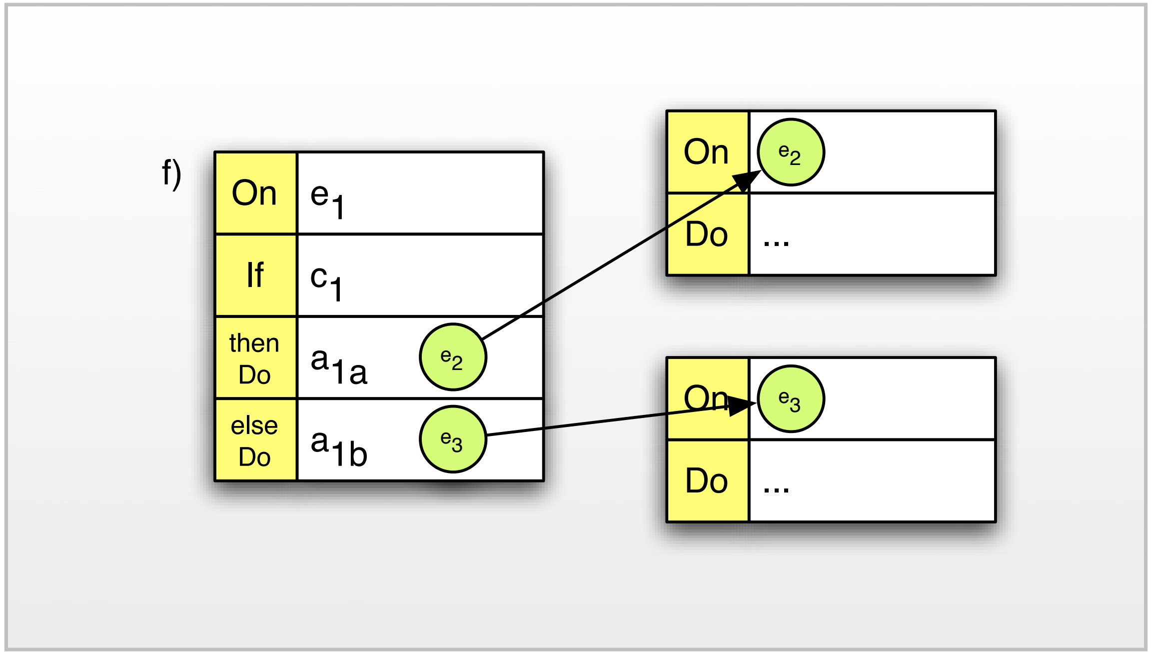
ECA Notation



ECA Sequence Flow



ECA Parallel Flow



ECA Choice

Mapping Workflow Activities to Agents

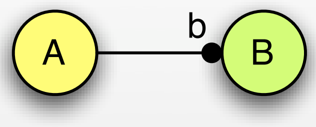
- Each workflow activity is mapped to a concurrent pi-calculus agent:
  - Each agent has pre- and post-conditions
  - Pre-condition = Event and Condition
  - Postcondition = Action

$$x.[a = b]\tau.\overline{y}.\mathbf{0}$$

Basic Activities in the Pi-Calculus

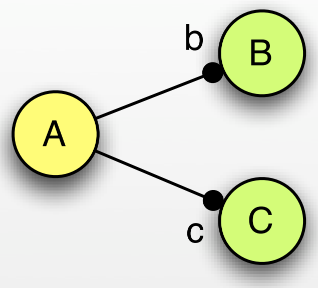
Basic Control Flow Patterns

- The basic control flow patterns capture elementary aspects of control flow



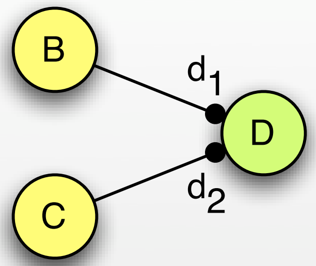
$$A = \tau_A.\bar{b}.\mathbf{0}$$
$$B = b.\tau_B.B'$$

Sequence



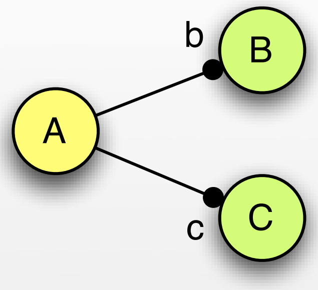
$$A = \tau_a.(\overline{b}.\mathbf{0}|\overline{c}.\mathbf{0})$$
$$B = b.\tau_B.B'$$
$$C = c.\tau_C.C'$$

Parallel Split



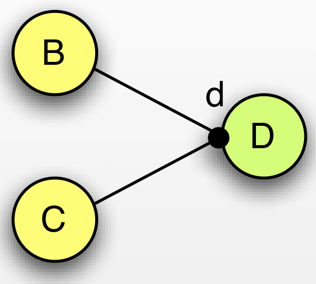
$$B = \tau_B.\overline{d_1}.\mathbf{0}$$
$$C = \tau_C.\overline{d_2}.\mathbf{0}$$
$$D = d_1.d_2.\tau_D.D'$$

Synchronization



$$A = \tau_A.(\overline{b}.\mathbf{0} + \overline{c}.\mathbf{0})$$
$$B = b.\tau_B.B'$$
$$C = c.\tau_C.C'$$

Exclusive Choice

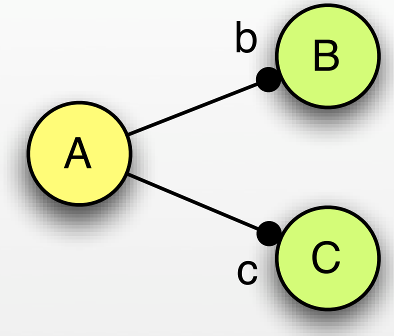


$$B = \tau_B.\overline{d}.\mathbf{0}$$
$$C = \tau_C.\overline{d}.\mathbf{0}$$
$$D = d.\tau_D.D'$$

Simple Merge

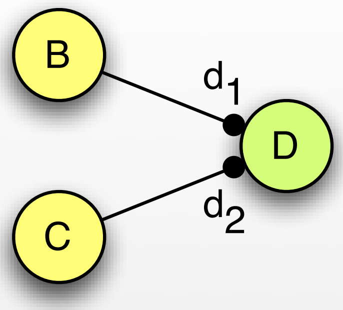
Advanced Branching and Synchronization Patterns

- The advanced branching and synchronization patterns require advanced concepts and map only partly to the basic activity template



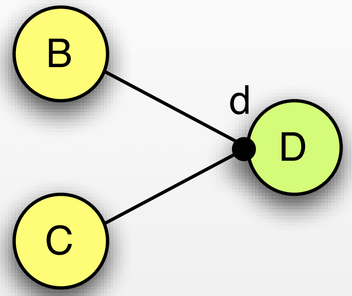
$$
\begin{aligned}
A &= (\mathbf{v}exec)\tau_A.(A_1|A_2)\\
A_1 &= \overline{exec}\langle b\rangle.\mathbf{0}+\\
&\quad\ \overline{exec}\langle c\rangle.\mathbf{0}+\\
&\quad\ \overline{exec}\langle b\rangle.\overline{exec}\langle c\rangle.\mathbf{0}\\
A_2 &= !exec(x).\overline{x}.\mathbf{0}\\
B &= b.\tau_B.B'\\
C &= c.\tau_C.C'
\end{aligned}
$$

Multiple Choice



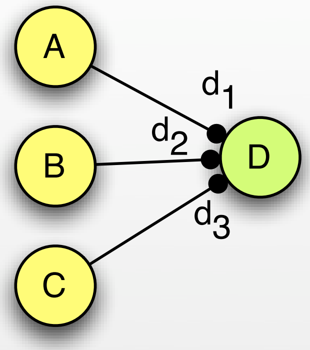
$$B = \tau_B.\overline{d_1}.\mathbf{0}$$
$$C = \tau_C.\overline{d_2}.\mathbf{0}$$
$$D = d_1.\tau_D.D' + d_2.\tau_D.D' + d_1.d_2.\tau_D.D'$$

Synchronizing Merge



$$B = \tau_B.\overline{d}.\mathbf{0}$$
$$C = \tau_C.\overline{d}.\mathbf{0}$$
$$D = !d.\tau_D.D'$$

Multiple Merge



$$A = \tau_A.\overline{d_1}.\mathbf{0} \quad B = \tau_B.\overline{d_2}.\mathbf{0} \quad C = \tau_C.\overline{d_3}.\mathbf{0}$$

$$D = (\mathbf{v}h, exec)(D_1|D_2)$$

$$D_1 = d_1.\overline{h}.\mathbf{0} \mid d_2.\overline{h}.\mathbf{0} \mid d_3.\overline{h}.\mathbf{0}$$

$$D_2 = h.\overline{exec}.h.h.D \mid exec.\tau_D.D'$$

Discriminator

$$D = (\mathbf{v}h, exec)((\prod_{i=1}^{m} d_i.\overline{h}.\mathbf{0}) \mid h.\overline{exec}.\{h\}_1^{m-1}.D \mid exec.\tau_D.D')$$

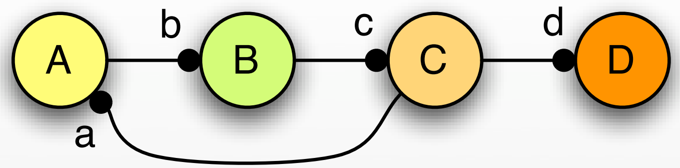
Discriminator Template

$$D = (\mathbf{v}h, exec)((\prod_{i=1}^{m} d_i.\overline{h}.\mathbf{0}) \mid \{h\}_1^n.\overline{exec}.\{h\}_{n+1}^m.D \mid exec.\tau_D.D')$$

N-out-of-M-Join Template

Structural Patterns

- Structural patterns show restrictions on workflow languages



$$A =!a.\tau_A.\overline{b}.\mathbf{0}$$
$$B =!b.\tau_B.\overline{c}.\mathbf{0}$$
$$C =!c.\tau_C.(\overline{a}.\mathbf{0} + \overline{d}.\mathbf{0})$$
$$D = d.\tau_D.D'$$

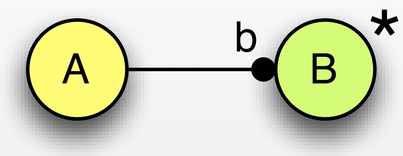
Arbitrary Cycles

Implicit Termination

- The implicit termination pattern terminates a sub-process if no other activity can be made active
  - Problem: Most engines terminate the whole workflow if a final node is reached
- The pi-calculus contains the final symbol **0**

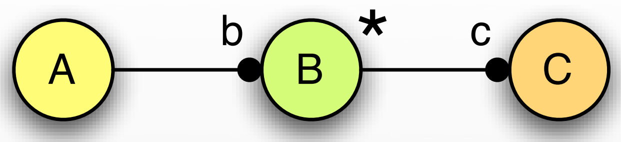
Multiple Instance Patterns

- Multiple instance patterns create several instances (copies) of workflow activities



$$A = \tau_A.!\overline{b}.\mathbf{0}$$
$$B = !b.\tau_B.B'$$

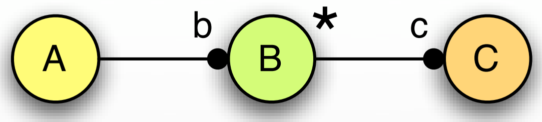
MI without Synchronization



$$A = \tau_A.\overline{b}.\overline{b}.\overline{b}.\mathbf{0}$$
$$B = !b.\tau_B.\overline{c}.\mathbf{0}$$
$$C = c.c.c.\tau_C.C'$$

$$A \mid B \mid C \equiv \tau_A.\{\overline{b}\}_1^n.\mathbf{0} \mid !b.\tau_B.\overline{c}.\mathbf{0} \mid \{c\}_1^n.\tau_C.C'$$

MI with a priori Design Time Knowledge



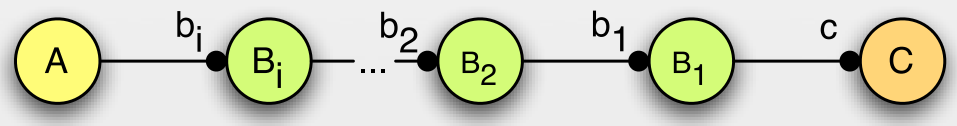
$$A = \tau_A . A_1(c)$$

$$A_1(x) = (\mathbf{v}y)\bar{b}\langle y\rangle . y\langle x\rangle . A_1(y) + \bar{x}.\mathbf{0}$$

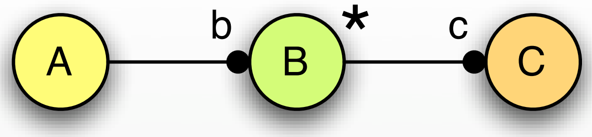
$$B = !b(y).y(x).\tau_B . y.\bar{x}.\mathbf{0}$$

$$C = c.\tau_C . C'$$

The pattern works like a dynamic linked-list:



MI without a priori Runtime Knowledge



$$A = (\mathbf{v}run)\tau_A.A_1(c) \mid run.!\overline{start}.\mathbf{0}$$

$$A_1(x) = (\mathbf{v}y)\overline{b}\langle y\rangle.y\langle x\rangle.A_1(y) + \overline{run}.\overline{x}.\mathbf{0}$$

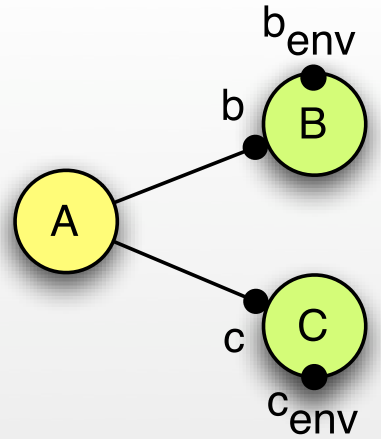
$$B = !b(y).y(x).start.\tau_B.y.\overline{x}.\mathbf{0}$$

$$C = c.\tau_C.C'$$

MI with a priori Runtime Knowledge

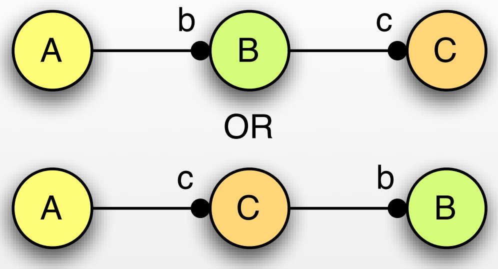
State-based Patterns

- State-based patterns capture implicit behavior of processes that is not based on the current case rather than the environment or other parts of the process



$$A = \tau_A.(\overline{b}.\mathbf{0}|\overline{c}.\mathbf{0})$$
$$B = b.(b_{env}.\overline{kill}.\tau_B.B' + kill.\mathbf{0})$$
$$C = c.(c_{env}.\overline{kill}.\tau_C.C' + kill.\mathbf{0})$$

Deferred Choice



$$A = \tau_A.\overline{x}.y.\overline{x}.y.A'$$
$$B = x.\tau_B.\overline{y}.\mathbf{0}$$
$$C = x.\tau_C.\overline{y}.\mathbf{0}$$

Interleaved Parallel Routing

$$A = check(x).([x = \top]\tau_{A1}.A' + [x = \bot]\tau_{A2}.A'')$$
$$B = M(\bot) \mid b.\overline{m}\langle\top\rangle.\tau_B.\overline{m}\langle\bot\rangle.B'$$
$$M(x) = m(x).M(x) + \overline{check}\langle x\rangle.M(x)$$

Milestone

Cancelation Patterns

- The cancelation patterns describe the withdrawal of one or more processes that represent workflow activities

$$A \mid \mathcal{E} \equiv a.\tau_A.A' + cancel.\mathbf{0} \mid !\tau_{\mathcal{E}}.\overline{cancel}.\mathbf{0}$$

Cancel Activity

Cancel Case

- The cancel case pattern cancels a whole workflow instance
- This is equal to Cancel Activity with the exception that all remaining processes receive a global cancel trigger

Data Representation

$$CELL \stackrel{def}{=} \nu c\ \overline{cell}\langle c\rangle.(CELL_1(\bot) \mid CELL)$$

$$CELL_1(n) \stackrel{def}{=} \overline{c}\langle n\rangle.CELL_1(n) + c(x).CELL_1(x)$$

Memory Cell

$$PAIR \stackrel{def}{=} \nu t\ \overline{pair}\langle t\rangle.(PAIR_1(\bot,\bot) \mid PAIR)$$

$$PAIR_1(m,n) \stackrel{def}{=} \bar{t}\langle m,n\rangle.PAIR_1(m,n) + t(x,y).PAIR_1(x,y)$$

Pairs, Tuples

$$STACK \stackrel{def}{=} \nu s\, \nu empty\ \overline{stack}\langle s, empty\rangle.(STACK_0 \mid STACK)$$

$$\begin{aligned} STACK_0 \stackrel{def}{=}\ & \overline{empty}.STACK_0 + s(newvalue).triple(next). \\ & \overline{next}\langle \bot, \bot, newvalue\rangle.STACK_1(next)\ , \end{aligned}$$

$$\begin{aligned} STACK_1(curr) \stackrel{def}{=}\ & curr(prev, test, value).(\overline{s}\langle value\rangle. \\ & ([test = \top]STACK_1(prev) + [test = \bot]STACK_0)+ \\ & s(newvalue).triple(next).\overline{next}\langle curr, \top, newvalue\rangle. \\ & STACK_1(next))\ . \end{aligned}$$

Stack

$$QUEUE \stackrel{def}{=} \nu q\, \nu empty\ \overline{queue}\langle q, empty\rangle.(QUEUE_0 \mid QUEUE)$$

$$\begin{aligned}QUEUE_0 \stackrel{def}{=} & \overline{empty}.QUEUE_0 + q(newvalue).triple(newtriple).\\ & \overline{newtriple}\langle \bot, \bot, newvalue\rangle.QUEUE_1(newtriple, newtriple)\end{aligned}$$

$$\begin{aligned}QUEUE_1(first, last) \stackrel{def}{=} & first(next, test, value).(\overline{q}\langle value\rangle.\\ & ([test = \top]QUEUE_1(next, last) + [test = \bot]QUEUE_0)+\\ & q(newvalue).triple(newtriple).\overline{newtriple}\langle \bot, \bot, newvalue\rangle.\\ & last(oldnext, oldtest, oldvalue).\overline{last}\langle newtriple, \top, oldvalue\rangle.\\ & QUEUE_1(first, newtriple)\ .\end{aligned}$$

Queue

$$I \stackrel{def}{=} s(x).\tau_I.I + empty.I'$$

Descructive Iterator

$$\nu\top\ \nu\bot\ S$$

$$TRUE = \overline{true}\langle\top\rangle.TRUE \qquad FALSE = \overline{false}\langle\bot\rangle.FALSE$$

Booleans

$$
\begin{aligned}
AND &\stackrel{def}{=} cell(v).and(b1, b2, resp).b1(x).b2(y).([x = \top][y = \top]\overline{v}\langle\top\rangle.AND_1 + \\
&\quad [x = \bot]\overline{v}\langle\bot\rangle.AND_1 + [y = \bot]\overline{v}\langle\bot\rangle.AND_1) \\
AND_1 &\stackrel{def}{=} (\overline{resp}\langle v\rangle.\mathbf{0} \mid AND)\ .
\end{aligned}
$$

Conjunction

$$OR \stackrel{def}{=} cell(v).or(b1, b2, resp).b1(x).b2(y).([x = \bot][y = \bot]\overline{v}\langle\bot\rangle.OR_1 + [x = \top]\overline{v}\langle\top\rangle.OR_1 + [y = \top]\overline{v}\langle\top\rangle.OR_1)$$

$$OR_1 \stackrel{def}{=} (\overline{resp}\langle v\rangle.\mathbf{0} \mid OR) \ .$$

Disjunction

$$NEG \stackrel{def}{=} neg(b, resp).true(t).false(f).b(x).(\\ ([b = t]\overline{resp}\langle false\rangle.\mathbf{0} + [b = f]\overline{resp}\langle true\rangle.\mathbf{0}) \mid NEG)$$

Negation

$$\langle \bot, \bot, \top, \bot, \top, \bot, \top, \bot \rangle$$

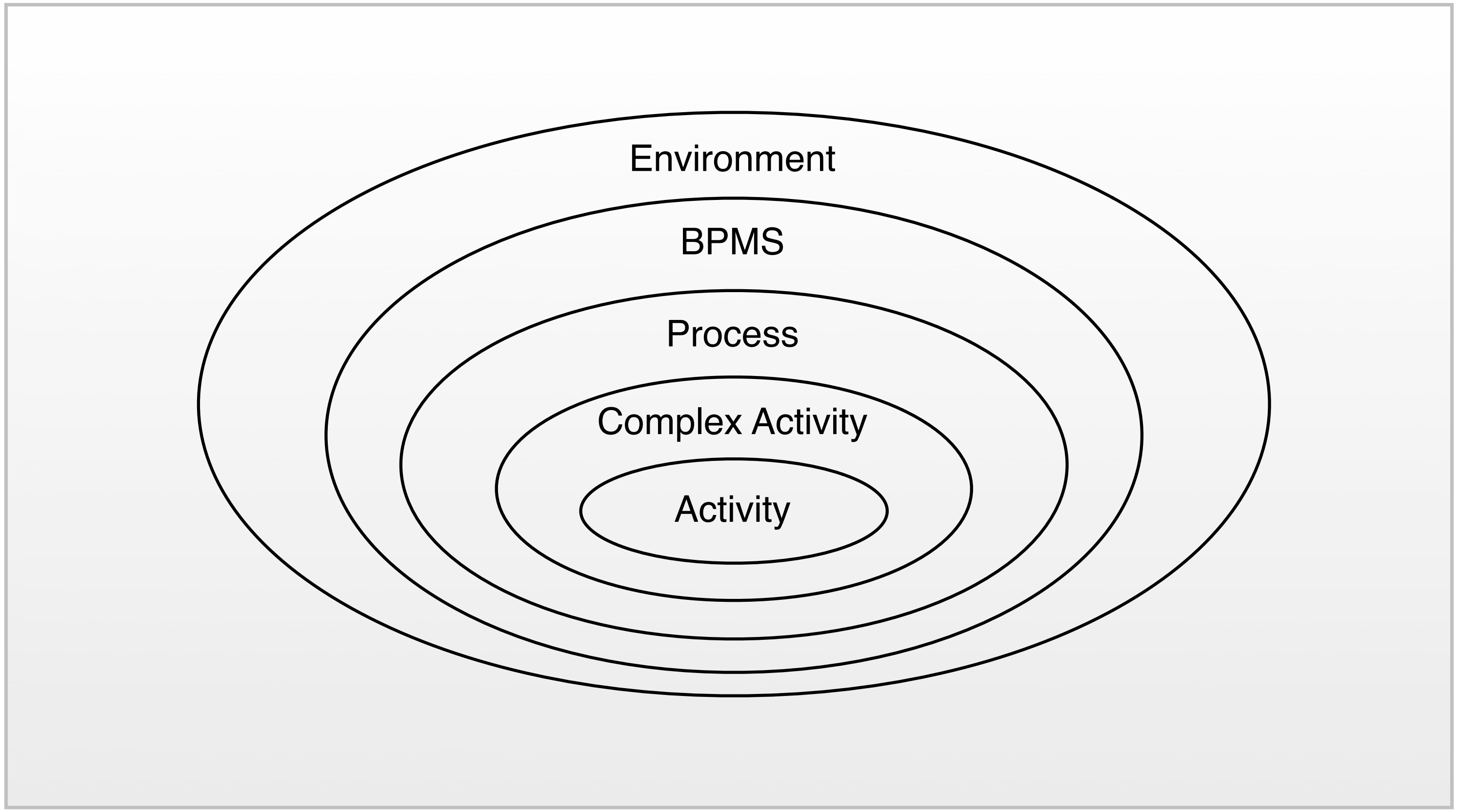
$$BYTE_{42} \stackrel{def}{=} \overline{byte42}\langle \bot, \bot, \top, \bot, \top, \bot, \top, \bot \rangle.BYTE_{42}$$

Bytes

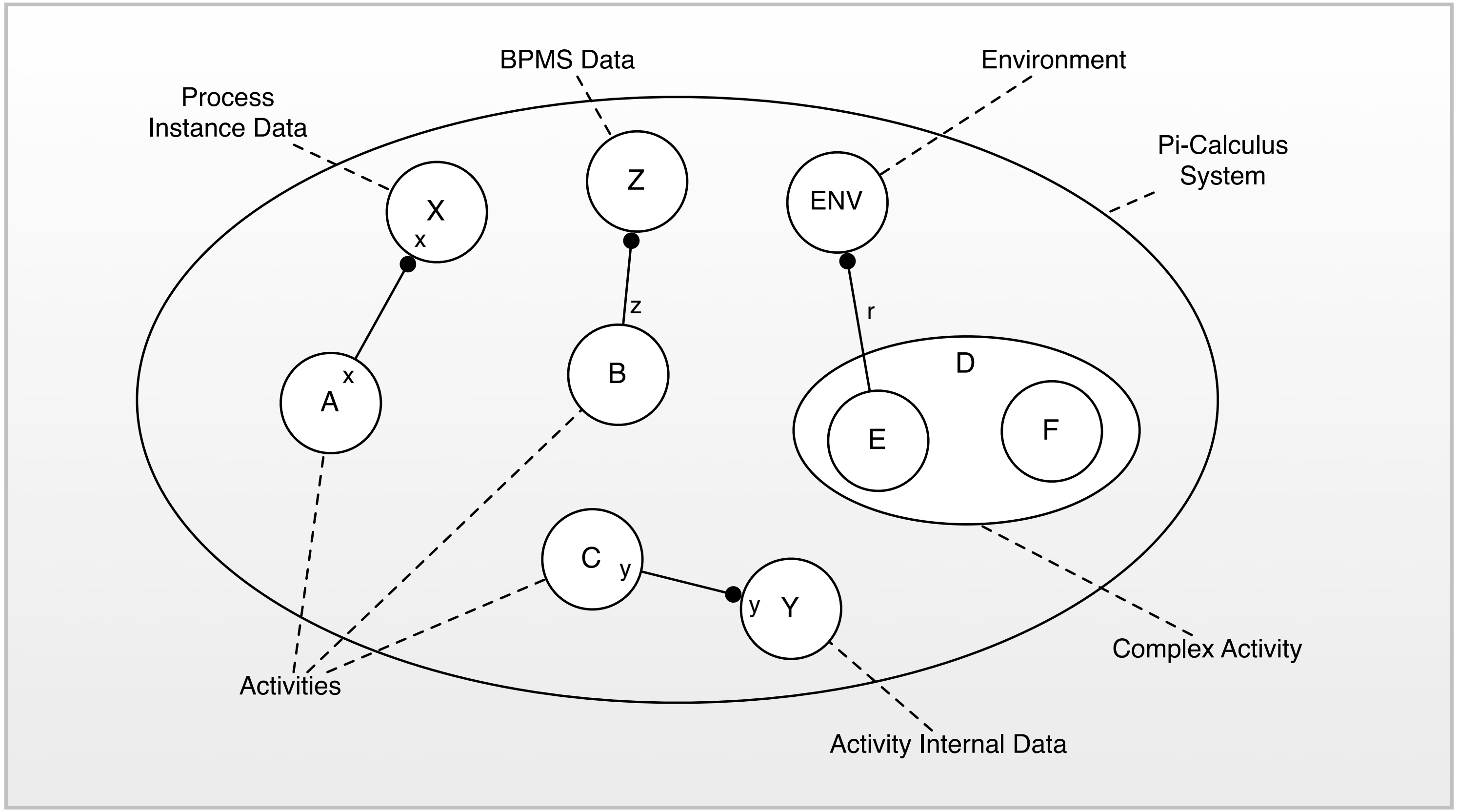
Further structures

- More structures are possible:
  - Natural numbers based on extended queues
  - Lists using natural numbers as indices (why?)
  - Strings
  - etc.

Workflow Data Patterns



Data Layers



Activities and Data

Some Sample Data Patterns

- Activity data
- Complex activity data
- Scope data
- BPMS data
- Data interaction: Activity to Activity
- Data interaction: Complex activities

Activity Data

- Data elements can be defined by activities which are accessible only within the context of individual execution instances of that activity:

$$A \stackrel{def}{=} \nu x \; cell(c).\tau.\mathbf{0}$$

Complex Activity Data

- Complex activities are able to define data elements, which are accessible by each of their components:

$$C \stackrel{def}{=} queue(q, e).(A \mid B)$$

Scope Data

- Data elements can be defined which are accessible by a subset of the activities in a process instance:

$$I \stackrel{def}{=} (A \mid B \mid \nu z\,(C \mid D))$$

BPMS Data

- Data elements are supported which are accessible to all components in each and every process instance and are within the control of the business process management system (BPMS):

$$BPMS \stackrel{def}{=} stack(s, e).(P_{enact}) \text{ and } P_{enact} \stackrel{def}{=} start.(P \mid P_{enact})$$

Data Interaction: Activity to Activity

- The ability to communicate data elements between one activity instance and another within the same process instance:

$$P \stackrel{def}{=} \nu d \; (cell(a).\tau.\overline{d}\langle a \rangle.\mathbf{0} \mid d(x).\tau.\mathbf{0})$$

Data Interaction: Complex Activities

- The ability to pass data elements to/from a complex activity:

$$C \stackrel{def}{=} d(x).(A \mid B)$$

$$C \stackrel{def}{=} \nu c1\ \nu c2\ (cell(u).\tau.\overline{c1}\langle u\rangle.\mathbf{0} \mid \nu v\ \tau.\overline{c2}\langle v\rangle.\mathbf{0} \mid c1(x).c2(y).\overline{d}\langle x, y\rangle.\mathbf{0})$$